Instructions on Using Git

1 Basics

Git is the current most popular version control system. It is similar to CVS in many ways, but there are some differences. Most importantly, each checked out Git repository has full version history by default and it allows local commits. Overall Git has more features and more flexibility than CVS. We convert the current CVS repositories to Git in order to provide a more effective and up-to-date version control environment to our users and developers. An effort is made to preserve the most important functionalities of the current CVS repository, including the ability to check out the whole SWMF as well as the stand-alone models.

1.1 Set up ssh connection to herot.engin.umich.edu and ssh config file (Mandatory)

To allow the Config.pl script accessing Git repositories, users are **REQUIRED** to add "herot" as an alias in the ~/.ssh/config file:

```
Host         herot                  # one can add extra names: herot swmfrepo
HostName     herot.engin.umich.edu  # very last time you type herot.engin.umich.edu
User         YOURUSERNAMEONHEROT    # only needed if different from local username
IdentityFile ~/.ssh/id_rsa          # this is probably not needed on most machines
```

Make sure that you do not need to type your password to access herot unless you want to type it hundreds of times. If you haven't done this for CVS, now you really should. Use

```
$ ssh-keygen                        # just hit return to all questions
$ ssh-copy-id herot                 # use password last time to copy key to herot
```

This should be done for all machines from which you want to access the Git repository.

2 SWMF Git Repository

The version control information is stored at the top level of the working directory which is different from the CVS system that stores information in each directory. To accommodate the SWMF structure where models can be checked out in stand-alone mode as well as part of the SWMF, we use multiple Git repositories. We created several Git repositories (SWMF.git, BATSRUS.git, GITM2.git, share.git, util.git and so on). All the SWMF related Git repositories are stored in the

herot:/GIT/FRAMEWORK/

directory. To make cloning (downloading) the SWMF and/or individual models, like BATS-R-US, easier, the *Config.pl* scripts can take care of cloning the Git repositories corresponding to the various subdirectories. In short, the SWMF git repository has multiple git repositories in its subdirectories.

3 Git Commands and Tools for Users

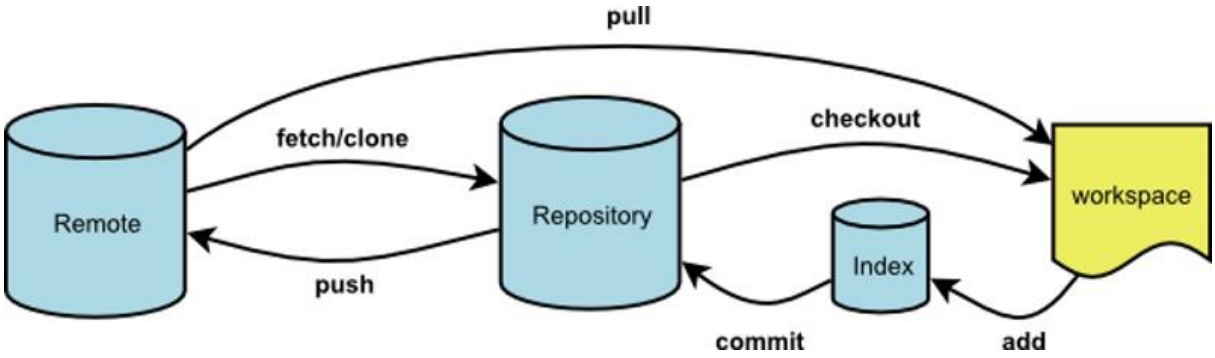


Figure 1 The working logic of Git system

3.1 Clone a remote git repository to a local one (similar to cvs checkout)

$ git clone SERVER_ADDRESS:ADDRESS/REPONAME LOCALNAME

The optional LOCALNAME allows changing the name locally (not available in CVS).

The command above clones the remote repository to the local machine with complete version history. This is an improvement over CVS because the history can be accessed without connecting to the remote server. However, the size of the local SWMF repository with all models included will be about 900 MB instead of the 187 MB size of the CVS repository. Using --depth=NUMBER allows limiting the version history, for example:

$ git clone --depth=1 herot:/GIT/FRAMEWORK/SWMF SWMF_NO_HISTORY

clones the SWMF with the latest version only and the size is reduced to about 260 MB. This reduces disk usage and initial download time.

To avoid typing the name of the Git server every time, it is convenient to define **gitclone** as an alias or function.

For csh and tcsh shells define a gitclone alias in the .cshrc file:
$ alias gitclone "git clone herot:/GIT/FRAMEWORK/\!*"

For sh, ksh, bash and zsh create a function in .profile, .bashrc, .zshrc or similar configuration file:
$ gitclone() { git clone herot:/GIT/FRAMEWORK/"$@"; }

With this definition, any Git repository can be cloned easily, for example

```
$ gitclone BATSRUS
$ gitclone SWMF --depth=1 SWMF_NO_HISTORY
```

3.2 Tools developed for the SWMF with many git repositories

The improved *Config.pl* scripts now checks the existence of the necessary components, for example, the share/ and util/ subdirectories, and clones them from herot if necessary. In addition, the *Config.pl* script in the SWMF will also clone the SWMF models (like BATSRUS, AMPS, etc) during installation as needed.

We provide a script *gitall* in *share/Scripts*. This script will recursively search the git repositories and execute the git command passed to it. It is best to link or copy gitall into the execution path, so it can be used easily from any directory. **Notice that gitall only searches the subdirectories**. For example, to check the status of all the git repositories in the SWMF type the following at the top-level SWMF directory:

```
$ gitall status
```

By default, there is output only if there is a change in the repository. Using the -v flag (verbose)

```
$ gitall status -v
```

will show all the git repositories even if there are no changes in them. Other possible uses include

```
$ gitall fetch origin          # get latest version of the code from herot
$ gitall difftool origin       # compare local version with the version copied from herot
$ gitall merge origin          # merge local and copied versions
$ gitall pull                  # simply update the local source code from herot
$ gitall push                  # push all the changes to herot, remember to commit first!
```

Doing commit with gitall is not recommend since commit logs are typically independent in different repositories.

3.3 Checking the difference between the local git repository and the remote server (similar to cvs -n update)

First of all, it is useful to define a visual difference tool, such as tkdiff, to be accessible by Git. Use the following command

```
$ git config --global diff.tool tkdiff
```

When other developers modified the project and pushed changes to the remote repository, it is useful to check the differences between the local and remote repositories. It can be done using gitall in top level of SWMF

```
$ gitall fetch origin                    # update the version information of the remote
$ gitall diff origin          # compare the local branch with 'origin' (the newest version in remote)
$ git difftool origin         # compare the local branch with origin using the difftool defined above
```

3.4 Update the git repository from the remote server (similar to cvs update)

If the 'origin' branch was downloaded with 'git fetch origin', the changes can be merged with the local master branch using

```
$ gitall merge origin
```

Please refer to the internet for information about dealing with conflicts during the merging. If you are confident about the correctness of the remote repository, the fetch and merge steps can be replaced with a single command:

```
$ gitall pull
```

This will update your local repository (equivalent to fetch+merge). This is the most common and simple approach.

4 Working with the SWMF in Git

4.1 Working with the entire SWMF

Clone the core SWMF repository without models (similar to **cvs checkout SWMF_core**):

```
$ gitclone SWMF
```

Now you have a local SWMF repository without share, util and the physics models. The new features of Config.pl can take care of cloning (downloading) the missing pieces:

```
$ ./Config.pl                          # show SWMF info and clone util and share if missing
$ ./Config.pl -install                 # install SWMF with all models (clone the entire SWMF)
$ ./Config.pl -install=BATSRUS,PWOM    # install SWMF, clone GM/BATSRUS, PW/PWOM if missing
$ ./Config.pl -install=AMPS            # reinstall SWMF, clone AMPS if necessary
```

Note that models that are already present will not be cloned again during reinstallation.

4.2 Working with stand-alone models (BATSRUS, GITM2, PWOM …)

First clone the BATSRUS repository (similar to **cvs checkout BATSRUS**):

$ gitclone BATSRUS

Like in the SWMF, now you can use Config.pl to get all the necessary repositories, for example

$ ./Config.pl -install -compiler=gfortran

will automatically clone (check out) the share, util and srcBATL repositories into BATSRUS if not yet present. Note that these are independent Git repositories.

4.3 Using SWMF_data and CRASH_data

Just like with the CVS version, the large files are stored in the SWMF_data and CRASH_data repositories. These can be checked out into the home directory as

$ cd
$ gitclone SWMF_data
$ gitclone CRASH_data

This should be done before installing the SWMF or stand-alone model so that the data/ symbolic links can be properly created.

The information provided by now should be sufficient for a user who will not change the code.

5 Git Commands and Tools for Developers

5.1 Settings for line endings

If you're using Git to collaborate with others, ensure that Git is properly configured to handle line endings, on macOS use

$ git config --global core.autocrlf

5.2 Make changes to local repository (similar to cvs add/remove/commit)

In essence use “git” followed by the Unix commands to remove and rename files and directories, use “add” to add a new file and “commit” to commit the changes into the local repository (note that CVS does not allow renaming files or removing directories and there is no local repository):

```
$ git rm -rf DIR1
$ git rm FILE1
$ git commit -m "removed FILE1 and DIR1 because …"
$ git mv FILE2 FILE3
$ git commit -m “renamed FILE2 to FILE3”
$ emacs FILE4
$ git add FILE4
$ git commit “Created FILE4 which does ….”
```

Without the -m flag the editor (defined by the $EDITOR environment variable) opens to allow logging the changes. Just like with CVS, meaningful logs that describe the changes, the reasons for them and the consequences are extremely important and useful. You can also commit files separately and provide separate logs. This is recommended if the changes in different files are not related to each other.

The steps above affect the local git repository only. This allows storing multiple versions with a complete version history locally without making changes in the remote repository.

5.3 Undo changes to local repository

Sometimes an accidental or temporary change is made. There are various ways to undo the change depending on the status of the change:

(1) Change hasn’t been added to the index (no git add or git commit has been done)

```
$ git checkout FILE1          # undo changes in FILE1
$ git checkout .              # undo all changes in the repository
```

(2) Change has been added to the index but not committed (git add but no commit)

```
$ git reset HEAD FILE1      # use git reset to remove changed FILE1 from index
$ git checkout FILE1        # undo changes on FILE1

$ git reset HEAD            # use git reset to remove all changed files from index
$ git checkout .            # undo all changes
```

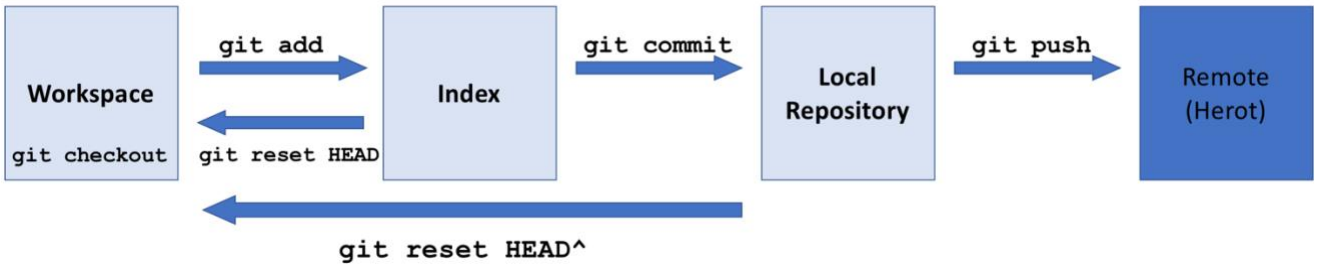
(3) Change has been committed but not pushed (git commit has been done)

```
$ git reset HEAD^           # revert the commit to current workspace
$ git checkout .            # undo all changes
$ git checkout FILE1        # undo changes on FILE1
```

(4) Change has been pushed (git push has been done)

Undo the modifications and commit and push them. Note that this really should not happen, because changes should be properly tested before being pushed to herot.

A figure to explain this:



5.4 Check the status of the present git repository (similar to cvs -n update -d)

After working for a while, to check what you have done, in top level of the SWMF type

$ gitall status

Check the changes in all the modified files:

$ gitall diff # list all differences in the repository tree
$ git difftool # after examining a file with the difftool, git jumps to the next file

Note: gitall difftool is not recommended, as it will open multiple tkdiff windows.
Also, tkdiff can be used directly to examine the changes made to FILE1:

$ tkdiff FILE1 # compare modified FILE1 to local master version
$ tkdiff -rorigin FILE1 # compare modified FILE1 to remote version fetched

5.5 Push changes to remote repository

After a period of local development, you may want to push the changes to the remote server Make sure you are on the master branch (same as **CVS HEAD**) in the local repository as well as in the submodules.

$ gitall push # Note that gitall executes in current directory and its subdirectories!

5.6 Get an old version of a git repository

Nightly tests of the SWMF can reveal bugs introduced during the code development. To identify the reason, it is necessary to compare versions of the SWMF at different dates. To get an old version, you need to clone the SWMF with full history. The default installation is without history

(to save download time and disk space). To get full history, use gitclone without the --depth parameter and install with the -history flag. The -date=DATE flag (which obtains full history and then sets the checkout date) can be used to get the code version of the required date:

```
$ gitclone SWMF SWMF_2018_06_19
$ cd SWMF_2018_06_19
$ ./Config.pl -clone -date="2018-06-19 19:00"
```

This will set the SWMF to the version tested at 19:00 EDT, June 19, 2018. Note that the resulting git repository will be in a status called “detached HEAD”. Changes made in this repository cannot be committed or pushed. This is similar to how CVS works when checked out for a given date.

One can revert a single file to an earlier version to correct an incorrect commit, for example. Note the backticks in the example below:

```
$ git reset `git rev-list -1 --before="2018-06-19 19:00" master` FILE1
$ git commit -m “revert FILE1 to an older version because…” FILE1      # commit to the local index
$ git checkout FILE1              # use the older version to overwrite the version in the workspace
$ git push                        # push the reverted version to herot if it works correctly
```

Note that the version overwritten remains available in the Git history (similar to CVS behavior).

6 The file permissions of a git repository

For a new git repository on herot, the ‘s’ (setuid) bit should be set for group users. When an executable is setuid, it runs as the user who owns the executable file instead of the user who invoked the program. The command to add the setuid, write and execute permissions for all the files in the git repository is:

```
$ chmod -R g+swX gitRepo
```